PARAMEDIC: PARALLEL METADATA ENVIRONMENT FOR DISTRIBUTED I/O AND COMPUTING

P. BALAJI, W. FENG, J. ARCHULETA, H. LIN, R. KETTIMUTHU AND R. THAKUR

Technical Report
Argonne National Laboratory #ANL/MCS-P1452-0807

ParaMEDIC: Parallel Metadata Environment for Distributed I/O and Computing*

P. Balaji[§†] W. Feng[¶] J. Archuleta[¶] H. Lin[‡]

R. Kettimuthu[§] R. Thakur[§] X. Ma[‡]

[§]Mathematics and Computer Science,
Argonne National Laboratory
{balaji, kettimut, thakur}@mcs.anl.gov

[¶]Dept. of Computer Science,
Virginia Tech
{feng, jsarch}@cs.vt.edu

[‡]Dept. of Computer Science,
North Carolina State University
{hlin2@unity, ma@csc}.ncsu.edu

Abstract

BLAST is a widely used software toolkit for genomic sequence search. mpiBLAST is a freely available, open-source parallelization of BLAST that uses database segmentation to allow different worker processors to search (in parallel) unique segments of the database. After searching, the workers write their output to a filesystem. While mpiBLAST has been shown to achieve high performance in clusters with fast *local* filesystems, its I/O processing remains a concern for scalability, especially in systems having limited I/O capabilities such as those using distributed filesystems spread across a wide-area network.

Thus, we present *ParaMEDIC*—an environment that decouples computation and I/O in distributed environments for applications such as mpiBLAST and dramatically reduces I/O overhead through metadata processing. Specifically, for mpiBLAST, ParaMEDIC partitions worker processes into compute and I/O workers. Compute workers, instead of directly writing output to the distributed filesystem, convert their output to metadata and send it to I/O workers. I/O workers, which physically reside closer to the actual storage, then process this metadata to re-create the actual output and write it to the filesystem. This approach allows ParaMEDIC to cut down on the I/O time, thus accelerating mpiBLAST by as much as 25-fold in some cases.

Keywords: mpiBLAST, I/O, distributed file-system

1 Introduction

Many computational biology tools and applications use nucleotide and protein sequence-searches to find similarities between different species of organisms. These searches enable biologists to find sibling species from a common ancestor. In 2003, sequence matching helped biologists to identify the similarities between the recent SARS virus and the more well-studied coronaviruses, thus enhancing the biologists' ability to combat the new virus.

Given the importance of sequence searches, researchers have designed a number of tools to perform sequence search in an efficient manner. Among the most widely used sequence-search tools is the Basic Local Alignment Search Tool (BLAST) from the National Center for Biotechnology Information (NCBI). BLAST, a multithreaded but sequential tool, identifies regions of local similarity between sequences. It compares a (nucleotide or protein) sequence query to a database of known sequences and calculates the statistical significance of the matches.

mpiBLAST [5, 7] is a freely available, open-source parallelization of NCBI BLAST. With more than 40,000 downloads in three years, mpiBLAST has become an integral component of many high performance cluster distributions [3, 4, 10, 11, 14, 15, 16, 17] and is an officially supported application at various high-performance computing facilities such as the System X [18] and the NSF TeraGrid [20]. The overall software architecture of mpiBLAST follows a master-worker model. The master fragments the sequence database across multiple nodes so that each fragment fits in memory. Each worker process then searches its unique portion of the database independently of the other workers. Once the search is complete, the results are merged and written out to a central file for later examination or post-processing by the biologist. While this model works well for clusters with a fast *local* filesystem, it does not work well when the filesystem is distributed across a wide-area network [8], which is the case in facilities such as the NSF TeraGrid and the distributed and encrypted compute environment between Argonne National Laboratory (ANL) and Virginia Tech (VT) over Internet2.

*This research is funded in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

†This author's work was also supported in part by the National Science Foundation Grant #0702182.

Figure 1: TeraGrid Architecture

NSF TeraGrid (Figure 1) is a distributed computing facility spread across various sites in the U.S. including the University of Chicago (Illinois), San Diego Supercomputing Center (California), Purdue University (Indiana), Texas Advanced Computing Center (Texas), and others. The San Diego Supercomputing Center (SDSC) also doubles up as a host for a global parallel filesystem (GPFS) that is visible and usable by all TeraGrid compute servers. All sites are connected using high-bandwidth (30 Gbps) optical links, but the large physical distance between the sites forces the latency to be high (several milliseconds) as well. Scientists use the computational power available at different locations to run searches and then write the final output to the globally shared filesystem to be viewed or postprocessed at a later time. While such a system provides good computational capability, for I/O rich applications, the distributed filesystem can form a significant bottleneck. For the distributed compute environment between ANL and VT, the case is even more severe because of a slower network link (1 Gbps) and encryption of data sent out over the wide-area network, which adds a substantial overhead.

Thus, in this paper, we propose *ParaMEDIC: **Para**llel **M**etadata **E**nvironment for **D**istributed **I**/O and **C**omputing*, a novel environment that decouples the computation and I/O for applications such as mpiBLAST in order to improve performance in distributed environments. Specifically, ParaMEDIC divides the worker processes into two groups: compute workers and I/O workers. The compute workers reside on the compute cluster (as they did earlier), while the I/O workers reside physically closer to the actual storage. For example, in the TeraGrid infrastructure, the compute workers can reside on any compute facility, while I/O workers always reside in SDSC.

When a biologist requests that a query be searched, the compute workers act in a manner almost identical to their typical approach. However, instead of directly writing the output to the filesystem, the workers process the output, generate *metadata* about the output and write the *metadata* to the filesystem. This metadata is several orders of magnitude smaller in size than the actual output, thus significantly reducing the I/O time taken by the compute workers. Once the metadata is written, the I/O workers perform a small amount of additional processing of the metadata to generate the final output and write it to the filesystem. Since the I/O workers are physically closer to the actual storage, writing the final output is substantially faster than having the compute workers write the final output.

We note that the total computation performed by ParaMEDIC is slightly higher than mpiBLAST because it first requires processing of the output to create metadata and then requires postprocessing of the metadata to regenerate the actual output at the I/O node. However, ParaMEDIC has the potential to significantly reduce the amount of I/O cost, especially in systems that have limited I/O capabilities. Thus ParaMEDIC trades a small amount of additional computation to dramatically reduce the I/O overhead.

Together with the detailed description of the new ParaMEDIC approach, this paper also presents evaluations of ParaMEDIC on three systems. The first system is a local cluster connected with a dedicated 10 Gbps network that also has the ability to vary both bandwidth and latency be-

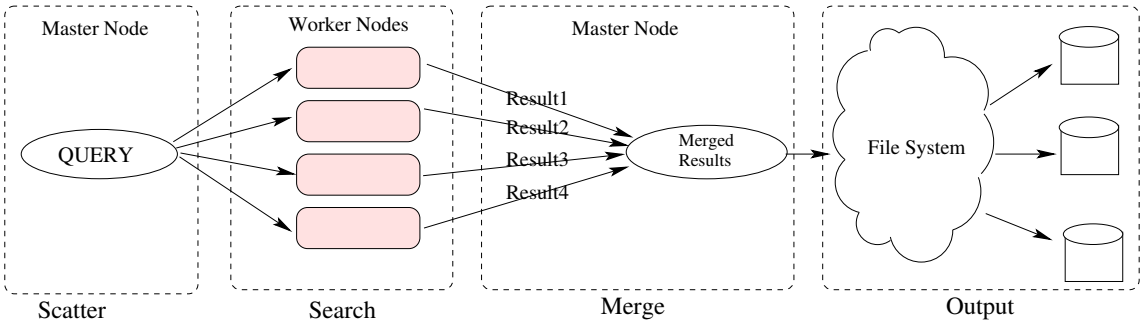


Figure 2: mpiBLAST Algorithm

tween any two nodes. We use this system to emulate various forms of high-latency and high-bandwidth distributed computing infrastructures. The second system is a secure encrypted filesystem hosted between ANL and VT over the Internet2 connection (1 Gbps network). This is a research infrastructure that is used to utilize compute resources from both sites in a convenient manner. The third system uses the University of Chicago and San Diego Supercomputing Center facilities of the TeraGrid infrastructure (30 Gbps network connectivity). Our experimental results demonstrate that ParaMEDIC achieves *order-of-magnitude improvements* in performance over the original mpiBLAST implementation in all three environments. Specifically, for the TeraGrid infrastructure we observed a 5-fold improvement and for the encrypted ANL-VT filesystem, we observed a 25-fold improvement in overall performance.

The remaining paper is organized as follows. In Section 2, we present a brief overview of mpiBLAST. We describe the ParaMEDIC approach to improve the I/O performance of mpiBLAST in Section 3. Experimental results on the local cluster as well as the various real systems are presented in Section 4. We present prior literature related to our work in Section 5 and the concluding remarks in Section 6.

2 Overview of mpiBLAST

In this section, we provide a brief overview of mpiBLAST. More details can be found in [5, 7].

With the size of the GenBank sequence databases doubling every 12 months [2, 9] and the computational horsepower of a single processor doubling only every 18–24 months, the growth rate of the databases has been fast outstripping the ability of a single processor to keep up. For instance, in 2002, searching for the 300-KB E. chrysanthemi in the NT database took 22.4 hours (80,775 seconds); by 2005, the same search but with an updated NT database took 49.1 hours (176,880 seconds) to complete. These trends motivated the development of a parallelized version of BLAST.

Because independent sequential queries can be run in parallel, the easiest (and trivial) way to parallelize BLAST is via *query segmentation*—each compute node searches its own unique query against its copy of the entire sequence database. However, the fact that entire sequence databases cannot fit into memory means that the operating system must swap pages to and from disk during the sequence search, thus adding a tremendous amount of overhead to the computation. This shortcoming led to an unconventional way of parallelizing BLAST—*database segmentation*, as done in mpiBLAST [5, 7].

Database segmentation fragments a database across multiple nodes to ensure that each fragment fits in memory. Then, the master node *scatters* an identical query to each worker node to *search* against its unique portion of the database, as shown on the left-hand side of Figure 2. Once the searches complete, the results are gathered and *merged* at the master process and the *output* written out to a central file, as shown on the right-hand side of Figure 2. The execution flow follows a *scatter-search-merge-output* pattern that is implemented within a *master-worker* parallelization model.

Database segmentation offers two primary advantages: (i) the time to distribute a database fragment is less than the time to distribute a complete copy of the database, and (ii) because the database fits in the aggregate memory of a parallel machine such as a cluster, superlinear speedups can be obtained due to the absence of disk I/O during the scatter, search and merge phases. However, disk I/O during the output phase when the result has to be written to a filesystem is inevitable. To this end, the performance of mpiBLAST heavily depends on the capability of the filesystem. Consequently, filesystems with limited I/O capabilities can significantly hamper the overall performance of mpiBLAST.

3 Design Overview

In this section, we present a detailed description of the ParaMEDIC framework. We first describe the overall framework in Section 3.1, followed by details of the generation of the metadata by the compute workers in Section 3.2. We then conclude the section with a description of how the postprocessing is performed by the I/O workers to regenerate the final output in Section 3.3.

3.1 The ParaMEDIC Framework

ParaMEDIC provides a two-tiered hierarchical framework for decoupling computation and I/O in mpiBLAST. The upper tier consists of two processes, *compute manager* and *I/O manager*, while the lower tier consists of two groups of processes – *compute workers* and *I/O workers*. The actual sequence search is handled by the compute workers. Once the output is generated, the compute master converts this output initially to raw metadata, and then processes and compresses it to form the final metadata. It then writes the final metadata to the filesystem and sends a signal to the I/O master. The I/O master, upon receiving a signal from the compute master, uses the I/O workers to process the metadata and generate the final output.

3.1.1 Trading Computation and I/O

As we can observe in the ParaMEDIC framework, the amount of computation required is higher than what is required by the original mpiBLAST implementation. For example, after the output is generated by the compute workers, it has to be processed to generate the metadata, sent to the I/O master, and again reprocessed by the I/O workers to regenerate the final output. However, such metadata processing potentially allows the the I/O cost to be significantly reduced. In other words, the ParaMEDIC framework aims at trading additional computation for reduced I/O cost.

The ParaMEDIC framework is very generic and tunable with respect to the amount of metadata processing required. The metadata can be a simple and basic compression of the actual output, a complex application-specific data structure or any other format that can be used to regenerate the final output. Obviously, depending on the kind of metadata that is to be generated, the metadata processing scheme will have a different amount of computational complexity and I/O overhead. For example, in a simple compression scheme, the amount of additional postprocessing required is minimal, but the I/O cost might still be high. On the other hand, with a complex application-specific data structure generation scheme, the amount of additional postprocessing required to regenerate the final output might be high, but the I/O cost will be very low. The right scheme to use will, of course, depend on the number of postprocessing compute resources available, as we will describe in Section 3.1.2.

3.1.2 Managing Compute and I/O Worker Processes

Managing the compute and I/O worker processes in ParaMEDIC essentially determines the tradeoff in the amount of time spent in computation versus the amount of time saved in I/O. In general, since the I/O worker processes are restricted to the cluster that hosts the filesystem, the number of I/O workers that are available is restricted. For example, in a distributed environment hosting 10,000 processors, only 1000 processors might reside on the same cluster that hosts the filesystem. Thus, in this case, it is ideal to maintain a 10:1 ratio between the number of compute workers and the number of I/O workers. Depending on the ratio, the appropriate metadata processing scheme needs to be picked as described in Section 3.1.1.

Irrespective of the scheme used, it is imperative that the ParaMEDIC approach use algorithms that can generate metadata from the result and regenerate the result from the metadata with minimal additional processing requirements. In other words, the framework has to ensure that the post-processing cost required to process the metadata and generate the final output is significantly less than the actual computation needed to search for the query sequence within the database.

In this paper, we demonstrate performance results only based on data structures specific to the mpiBLAST application. Other schemes including basic data compression are deferred to future work.

3.2 Metadata Processing for mpiBLAST

Several sequence databases use unique identifiers for each sequence in the database. For the nucleotide database, for example, GenBank Identifiers (GIs) are used to represent the different sequences. These GIs, though unique for each sequence, are not ordered in the database. The result that is generated from mpiBLAST typically consists of the sequences themselves, together with a significant amount of additional information such as details about the software package, additional information about each sequence, pattern matches that BLAST discovered for each sequence, the e-value representing how close the match was for each sequence, and various other statistics.

ParaMEDIC parses through the results generated by the mpiBLAST compute workers and extracts the GI information for each matching sequence in the database. Once the GI information for all the matching sequences in the database for each query sequence is identified, this data is compressed by discarding duplicate GI values (for example, when two query sequences match the same sequence in the database) and sent to the I/O workers. Depending on the query, this compressed GI information can be nearly three to four orders of magnitude smaller than the actual output.

We note that as the number of query sequences increase or as the number of result sequences required by the user increases, the metadata generated as well as the time for generating and processing the metadata increases. Thus, at some stage, the additional cost of metadata processing becomes greater than the saving in I/O time that is achievable. In order to address this, ParaMEDIC finds the size of the metadata that is created and uses a tunable threshold parameter to determine whether decoupled computation and I/O can provide any benefit. If the metadata size is larger than the threshold, the scheme falls back to the regular mpiBLAST implementation.

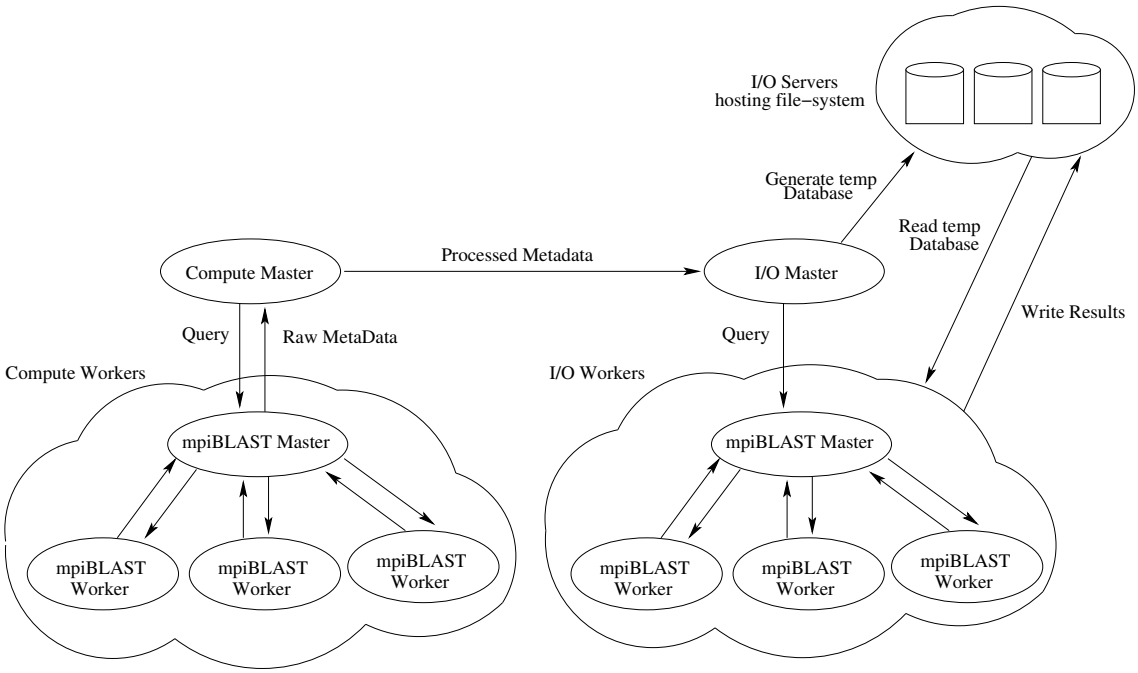


Figure 3: ParaMEDIC Framework

3.3 I/O Post-Processing

I/O postprocessing is handled by the I/O workers in ParaMEDIC. The postprocessing is comprised of two primary components—database creation and query search. In the first component, the I/O master creates a new temporary database based on the *matched segments* that were found by the compute workers. This temporary database is typically *much* smaller than the original database. For example, in the default configuration, if a single query sequence is provided by the user, while the actual database has about 5 million sequences the temporary database will have at most 500 sequences, i.e., 0.01% of the original size. Once the temporary database is created, the I/O workers recompute the original query sequences against this temporary database to generate the final output. Since the temporary database is typically very small, this search time is minimal.

The size of the metadata generated directly impacts the size of the temporary database. Thus, as the number of unique sequences that match the query sequences increases (either as the number of query sequences increases or the number of matches requested by the user increases), the size of the temporary database, and consequently, the I/O postprocessing time increases. Again, the threshold (mentioned in Section 3.2) ensures that the metadata size in this scheme is small and thus the ParaMEDIC scheme beneficial.

4 Performance Evaluation

This section presents a performance evaluation of ParaMEDIC-enhanced mpiBLAST (hereafter referred to as simply ParaMEDIC) and compares it with the existing mpiBLAST implementation. Specifically, we evaluate the two schemes in a local cluster emulating various distributed-computing infrastructures in Section 4.1. Evaluations on a distributed testbed between Argonne National Laboratory and Virginia Tech over the Internet2 network connection are presented in Section 4.2. Evaluations on the TeraGrid infrastructure with nodes from the University of Chicago and San Diego Supercomputing Center participating in the experiment are presented in Section 4.3.

All experiments have been performed with the Nucleotide (NT) database, which is larger than 20GB and contains over 5 million sequences.

4.1 Local Cluster Evaluation

In this section, we compare the performance of the ParaMEDIC scheme with that of the existing mpiBLAST implementation on a local cluster, which emulates various distributed-computing infrastructures.

4.1.1 Experimental Testbed for the Local Cluster

The testbed used in this configuration consists of 24 nodes, each equipped with dual-processor, dual-core (4 cores) Opteron 2220 2.8 GHz processors. Each processor has 2 MB of L2-cache (1 MB local L2 cache for each core). The nodes were equipped with 4 GB of 667-MHz DDR2 SDRAM and four SATA disks using a software RAID0. The network used to connect these machines was the NetEffect NE010 10-Gigabit Ethernet iWARP adapters. For the experiments, however, they were used as regular 10-Gigabit Ethernet adapters with host-based TCP/IP without using the iWARP support. The nodes were installed with Fedora Core

6 Operating System. To emulate the various distributed-computing infrastructures, we used the NetEm software package [1]; this package allows data from a network to be delayed without hampering the overall throughput, thus emulating high-latency, high-bandwidth distributed-computing environments.

The default configuration used in this section is a 50KB query file, a network delay of 50ms, 80 worker processes and 500 output sequences. We vary each of these parameters individually and keep the remaining parameters constant (assigned to their default configuration).

4.1.2 High-Bandwidth, High-Latency Networks

In this section, we analyze the impact of distributed environments connected with high-latency, high-bandwidth networks on the performance of ParaMEDIC and basic mpiBLAST. For this experiment, we divide the local cluster into two logical subclusters. While all the nodes are connected with a 10 Gbps network, we artificially delay the communication between nodes belonging to different subclusters. The socket buffer size used is set to at least be equal to the bandwidth-delay product of the network so as to maximize the performance the network subsystem can provide. Four nodes in the second subcluster (each node with 4 SATA disks configured as a software RAID0) host a PVFS2 filesystem visible to all nodes in both the subclusters. This setup represents a scenario similar to the TeraGrid where two clusters are geographically distributed, and the second cluster also doubles up to host a filesystem that is visible from both clusters.

For the evaluation, both ParaMEDIC and mpiBLAST use 20 dual-processor, dual-core nodes hosting a total of 80 worker processes. For mpiBLAST, all these processes are hosted in the first subcluster. For ParaMEDIC, however, the worker processes are divided into 76 compute workers (which are hosted on the first subcluster) and 4 I/O workers (which are hosted on the second subcluster)—a 19:1 ratio of compute to I/O workers.

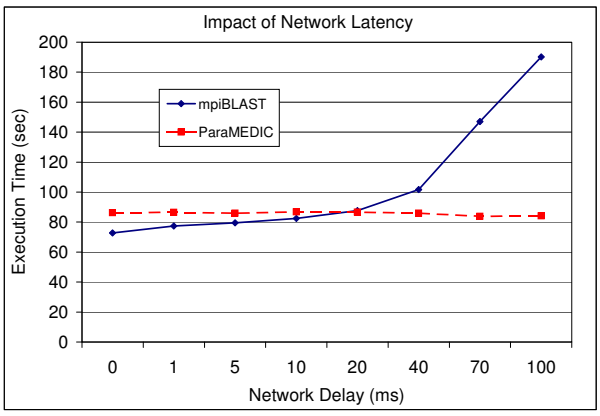


Figure 4: Impact of High-Bandwidth High-Latency Networks

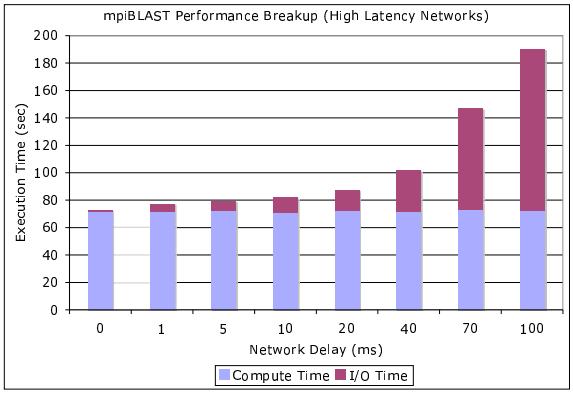
As shown in Figure 4, when the network delay between the two subclusters is low, the original mpiBLAST outperforms ParaMEDIC. This result is expected because of two reasons: (i) ParaMEDIC performs additional computation for converting the search results to metadata and converting the metadata back to the final output and (ii) the number of compute resources that ParaMEDIC uses for the actual computation is only 76 cores, as opposed to the 80 cores used by mpiBLAST—the remaining 4 cores in ParaMEDIC are used for the postprocessing. As the network delay increases, however, ParaMEDIC starts to outperform mpiBLAST. In fact, for a network delay of 100 ms, ParaMEDIC outperforms mpiBLAST by a factor of 2.26. This improvement in performance is attributed to two factors. First, the high network latency causes degradation in the filesystem operations that are required whenever data needs to be written or read from the server. Second, the total amount of data written in mpiBLAST over the I/O subsystem is much higher as compared to ParaMEDIC, since ParaMEDIC writes only metadata that is significantly smaller than the final results to the filesystem.

To further understand these results, we show the performance breakdown of the time taken by mpiBLAST and ParaMEDIC in Figure 5. As shown in Figure 5(a), for mpiBLAST, as the network delay increases, the I/O time increases very quickly. Thus, though the computation time does not change much, the overall execution time suffers. On the other hand, for ParaMEDIC (Figure 5(b)), the computation time, the I/O time, and the postprocessing time required to handle the metadata are fairly constant for all values of network delays. This result is anticipated because the only component in ParaMEDIC that would be affected by the network latency is the postprocessing, since it requires moving the metadata from the compute workers to the I/O workers. And because the metadata amount is very small (few KB), this time typically does not make any difference to either the postprocessing time or the overall execution time of the application.

4.1.3 Varying the Number of Worker Processes

This section analyzes the performance of mpiBLAST and ParaMEDIC with varying numbers of worker processes. The total number of worker processes allotted to each scheme is the same. For ParaMEDIC, however, four of these processes are always allocated as I/O workers in our experiments. In other words, in this section, we study the impact of the different ratios for the compute workers to I/O workers on the performance of ParaMEDIC.

Figure 6 shows the performance of the two schemes with varying number of worker processes with a constant network delay of 50ms. As shown in the figure, when the number of worker processes is high, ParaMEDIC outperforms mpiBLAST. However, as the number of worker processes decreases, the performance of ParaMEDIC degrades



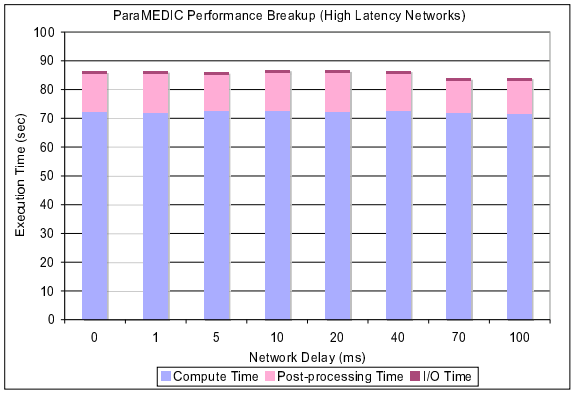
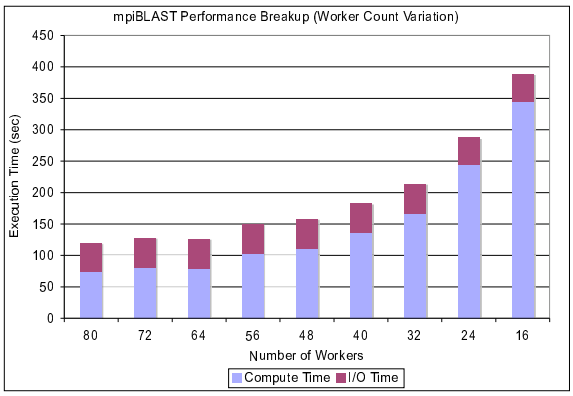


Figure 5: Breakup of Performance with Network Delay: (i) mpiBLAST and (ii) ParaMEDIC



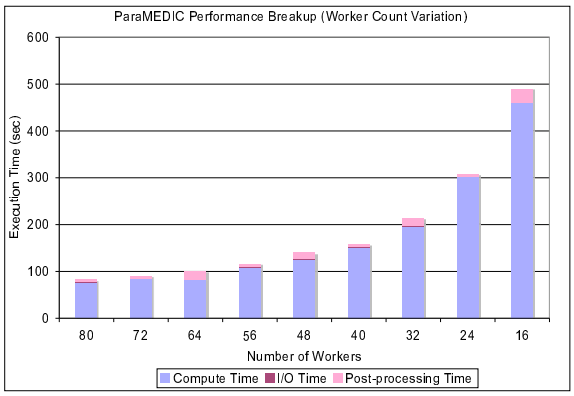


Figure 7: Breakup of Performance with Varying Number of Worker Processes: (i) mpiBLAST and (ii) ParaMEDIC

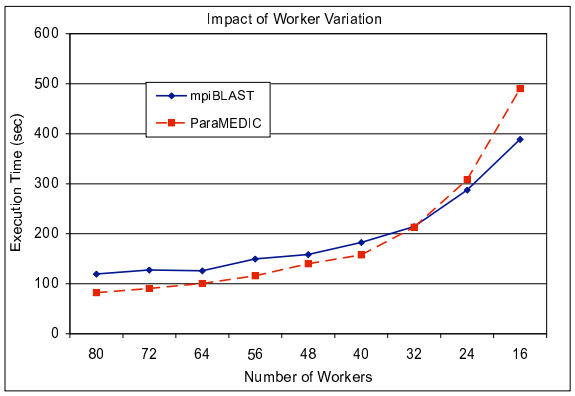


Figure 6: Varying the Number of Worker Processes

faster than that of mpiBLAST. For less than 24 worker processes (5:1 ratio of compute-to-I/O workers), we see that mpiBLAST outperforms ParaMEDIC.

This behavior is related to the number of compute workers each scheme has for performing the actual sequence search, as illustrated in Figure 7. Specifically, when the total number of worker processes available is N, ParaMEDIC uses only (N-4) of them for the sequence search. The remaining 4 processes are used for I/O processing. Thus, when N is very large, the increase in computation time caused by using lesser workers (approximately N / (N - 4)) is not very high. However, when N is small, the increase in computation time can be substantial. For example, when N is 8 processes, ParaMEDIC uses only 4 processes for the sequence search while mpiBLAST uses 8. Thus, the computation time taken by ParaMEDIC is nearly twice that of mpiBLAST. This overshadows any benefit in the I/O time ParaMEDIC can bring about, causing it to deliver worse performance than mpiBLAST.

This behavior is, of course, tunable by using different metadata processing schemes as described in Section 3.1.1. However, as described earlier, in this paper we only study the performance based on data structures specific to the mpiBLAST application. Other schemes (such as basic compression), which require lesser postprocessing and hence can deal with smaller compute-to-I/O worker ratios, are deferred to future work.

4.1.4 Varying the Number of Sequences Requested

In this section, we vary the number of sequences requested (both input query and output result sequences) and study

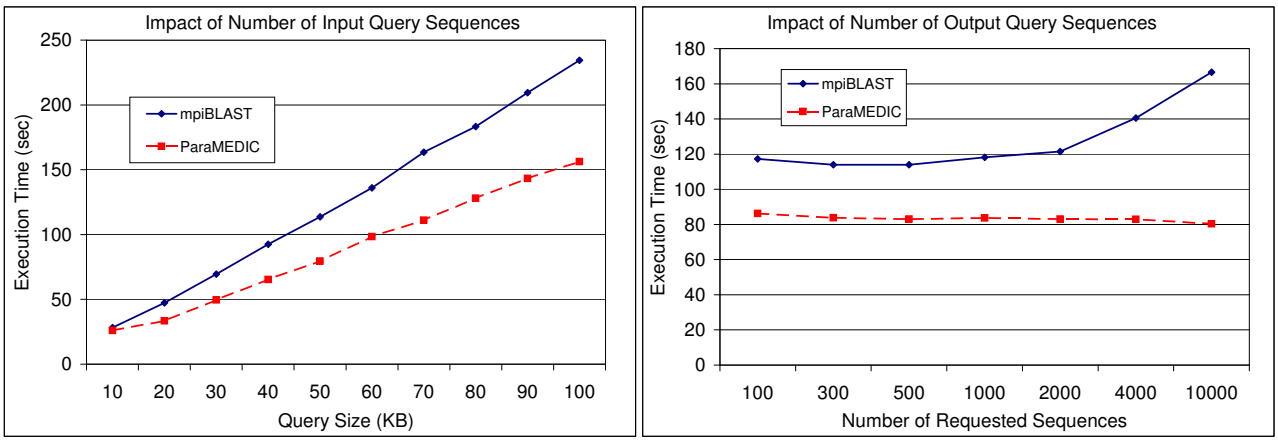


Figure 8: Varying the Number of Requested Sequences: (i) Input Query Sequences and (ii) Output Result Sequences

their impact on the performance of the two schemes. Varying the number of sequences in the input query increases the search time for both mpiBLAST and ParaMEDIC. However, it is also expected to impact the postprocessing time for ParaMEDIC, thus affecting it more than mpiBLAST. At the same time, an increase in the input query size also typically results in more output. This, on the other hand, can potentially impact mpiBLAST more than ParaMEDIC.

Figure 8(a) shows the performance of the two schemes with increasing input query sizes. We see that while the increase in the input query size increases the execution time of ParaMEDIC, it has a more dramatic effect on mpiBLAST. Thus, as the query size increases, the performance difference between the two schemes increases, with ParaMEDIC outperforming mpiBLAST by about 66% for a 100KB query file size.

Figure 8(b) shows the impact of increasing the number of requested output result sequences. The number of output result sequences does not impact the computation much, but can affect the amount of I/O. Thus, because the I/O cost for ParaMEDIC is very low, its performance does not vary much. On the other hand, since the I/O cost for mpiBLAST is very high, its performance is affected significantly.

4.1.5 Impact of Encrypted Filesystems

For distributed filesystems that span unsecure network connections (such as the Internet), data encryption is commonly used to protect transmitted data in several environments such as government national laboratories and other secure facilities. Figure 9 shows the impact of such data encryption on the performance of the two schemes. As shown in the figure, the performance of the two schemes is similar to the case where there is no file encryption, except that the performance of mpiBLAST degrades faster. This is attributed to the data encryption overhead. That is, since all the data that is being transmitted has to be encrypted and the amount of data transmitted by mpiBLAST over the unsecure network is significantly larger than ParaMEDIC, encryption affects mpiBLAST more significantly as compared to ParaMEDIC.

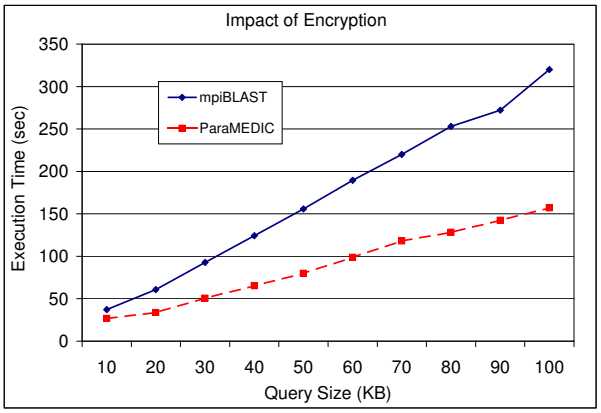


Figure 9: Impact of Encrypted Filesystems

4.2 Distributed Setup between ANL and VT

In this section, we evaluate the performance of mpiBLAST and ParaMEDIC on a distributed system between ANL and VT connected over Internet2. Since the network connecting the two clusters is *not* secure, data encryption is used to protect the data transmitted.

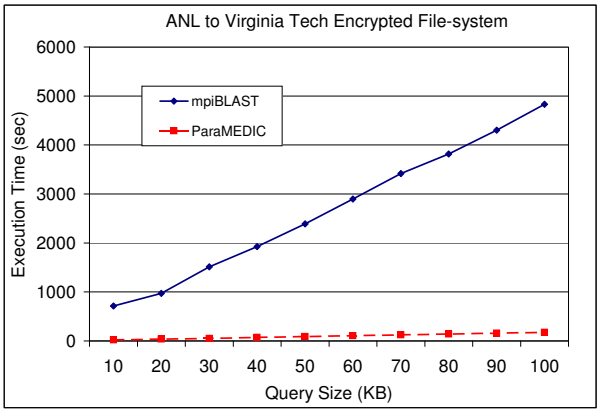


Figure 10: ANL-VT Encrypted Filesystem

As shown in Figure 10, ParaMEDIC significantly outperforms mpiBLAST in this environment. Further, as the query size increases, the performance difference between the two schemes increases. For a query size of 100 KB, we observe more than a *25-fold improvement* in performance for ParaMEDIC as compared to mpiBLAST. This difference is attributed to multiple aspects. First, given the shared network connection between the two sites, the effective network performance achievable is usually lower than within the cluster. Thus, with mpiBLAST transferring the entire output result over this network, its performance would be heavily impacted by the network performance. Second, since data communicated is encrypted, mpiBLAST also has to pay the penalty for such encryption. Though ParaMEDIC also pays such data encryption penalty, the amount of data it transfers is significantly lesser, and hence the penalty is less as well. Third, the distance between the two sites causes the communication latency to be high. Thus, filesystem operations tend to take a large amount of time, resulting in further loss of performance.

4.3 TeraGrid Infrastructure

The TeraGrid infrastructure represents a widely used real environment for several compute- and I/O-intensive applications including mpiBLAST. As described in Section 1, a GPFS-based distributed filesystem is hosted at SDSC, which can be accessed from all facilities, and forms a part of the TeraGrid facility. Since TeraGrid is a dedicated facility, it does not utilize any encryption of the data exchanged between sites.

For the experiments in this section, we utilized the nodes at the University of Chicago and SDSC. The I/O workers in ParaMEDIC are always scheduled at SDSC because of its close proximity to the actual storage. The compute workers, however, are scheduled at the University of Chicago.

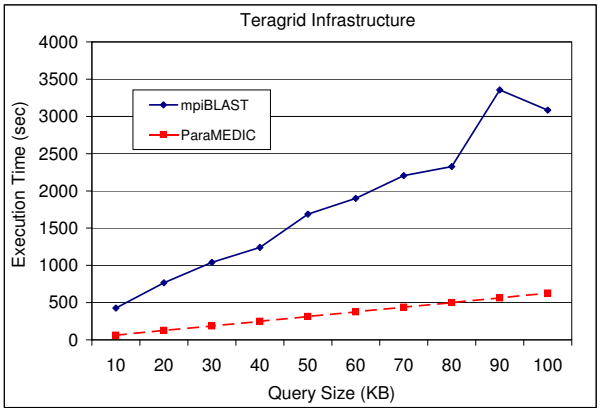


Figure 11: Evaluation on the TeraGrid Infrastructure

Figure 11 illustrates the performance of mpiBLAST and ParaMEDIC on the TeraGrid infrastructure. While the final output is written to the same global filesystem in both cases, mpiBLAST suffers from the fact that the compute workers are performing the I/O for the output results. Since they reside on a remote cluster as compared to the actual storage, their I/O performance is limited resulting in an overall degradation in execution time. For ParaMEDIC, on the other hand, since the I/O workers are performing the I/O for the output results, the amount of time taken is significantly smaller. For a query file size of 100 KB, ParaMEDIC outperforms mpiBLAST by a factor of about *five times*.

Figure 12 shows the performance breakdown of the two schemes. As shown in the figure, as the query size increases, the computation time for both mpiBLAST and ParaMEDIC increases. However, for mpiBLAST, the I/O time also increases very quickly while for ParaMEDIC there is essentially no difference in the I/O time with increasing query size. This result shows that ParaMEDIC is only minimally impacted by the limited I/O of the subsystem and that it efficiently distributes its worker processes across the system to achieve high performance.

5 Related Work

Alleviating the I/O bottleneck in parallel BLAST has been a subject of ongoing research. pioBLAST, is a recent work by our group that offloads I/O from the master and distributes it among the workers [12]. By having each worker write a portion of the output file in parallel, the I/O time is reduced. This, however, is efficient only when the filesystem can effectively handle concurrent parallel writes. For distributed filesystems where the inter-cluster communication is a bottleneck (due to high latency and/or limited bandwidth), however, this is not as beneficial. Further, the synchronization operations of a parallel filesystem will hinder performance for filesystems connected across a WAN.

ScalaBLAST [13] is another genome sequence search tool that is optimized for distributed as well as shared-memory systems. Though this paper uses ParaMEDIC with mpiBLAST, the concept is generic and can be used with other sequence search tools, including ScalaBLAST, as well.

Outside the realm of parallel BLAST, two other approaches to decoupling computation and I/O are MapReduce and TCP Linda. MapReduce is a programming model for processing and generating large datasets [6]. TCP Linda is a virtual shared-memory system whereby parallel processes execute simultaneously and exchange data by generating, reading, and consuming data objects [19]. Both MapReduce and TCP Linda decouple the different phases of an algorithm and transfer intermediate objects to the appropriate processes. However, unlike ParaMEDIC, neither of these approaches does any conversion of output to balance the I/O time with additional computational requirements.

In summary, the ParaMEDIC framework derives a number of insights from existing literature and extends them in a

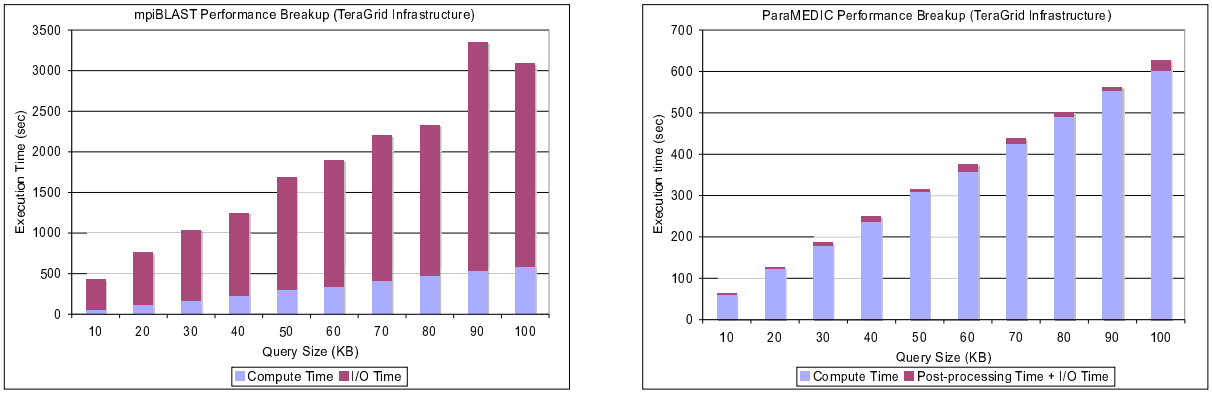


Figure 12: TeraGrid Infrastructure Performance Breakup: (i) mpiBLAST and (ii) ParaMEDIC

novel and interesting manner to achieve high performance for I/O rich applications such as mpiBLAST.

6 Conclusions and Future Work

mpiBLAST is an open source, freely available parallelization of BLAST, a widely used software for genome sequence searching. In spite of several previous enhancements, I/O processing in mpiBLAST is still a concern, especially in environments that use distributed filesystems with limited I/O capabilities. In this paper, we presented *ParaMEDIC*—an environment that decouples computation and I/O in applications such as mpiBLAST and dramatically reduces I/O costs using metadata processing. Specifically, for mpiBLAST, ParaMEDIC separates the worker processes into compute and I/O workers and schedules I/O workers physically closer to the actual storage. The compute workers, instead of directly writing the entire result to the filesystem, write metadata information corresponding to the output. The I/O workers use this metadata to recompute the final output results and write it to the filesystem. We have demonstrated that ParaMEDIC can achieve several-fold improvement compared to mpiBLAST in some cases.

As future work, we would like to further generalize the ParaMEDIC framework to allow multiple metadata formats with different tradeoffs for additional computation and reduced I/O. We also plan on studying the metadata patterns that are used by other applications as well.

References

[1] NetEm: Network Emulator. http://linux-net.osdl.org/index.php/Netem.

[2] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler. GenBank. *Nucleic Acids Res.*, 30:17–20, 2002.

[3] BioBrew / NPACI Rocks. `http://bioinformatics.org/biobrew/`.

[4] Cray. `http://www.cray.com/solutions/life/applications.html`.

[5] A. Darling, L. Carey, and W. Feng. The Design, Implementation, and Evaluation of mpiBLAST. In *International Conference on Linux Clusters: The HPC Revolution 2003*, 2003.

[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[7] W. Feng. Green destiny + mpiblast = bioinfomagic. In *International Conference on Parallel Computing (ParCo)*, 2003.

[8] M Gardner, W Feng, J Archuleta, H Lin, and X Ma. Parallel genomic sequence-searching on an ad-hoc grid: Experiences, lessons learned, and implications. In *ACM/IEEE SC2006: The International Conference on High-Performance Computing, Networking, and Storage*, 2006.

[9] Gold - Genomes Online Database. `http://www.genomesonline.org/`.

[10] IBM BlueGene. `http://researchcomp.stanford.edu/hpc/archives/BlueGene.pdf`.

[11] iNquiry. `http://www.bioteam.net/`.

[12] H. Lin, X. Ma, P. Chandramohan, A. Geist, and N. Samatova. Efficient Data Access for Parallel BLAST. In *International Parallel and Distributed Processing Symposium*, Apr 2005.

[13] C. Oehmen and J. Nieplocha. ScalaBLAST: A Scalable Implementation of BLAST for High-Performance Data-Intensive Bioinformatics Analysis. *IEEE Trans. Parallel Distrib. Syst.*, 17(8):740–749, 2006.

[14] Orion Multisystems. `http://www.orionmulti.com/support/faq_mpiblast`.

[15] Penguin Computing / Scyld. `http://bioinformatics.org/biobrew/`.

[16] Rocketcalc. `http://www.rocketcalc.com/package.php?Key=15`.

[17] Scalable Informatics. `http://www.scalableinformatics.com`.

[18] Terascale Computing Facility. `http://www.tcf.vt.edu/`.

[19] TCP Linda. `http://www.lindaspaces.com/products/linda_overview.html`.

[20] Teragrid. `http://www.teragrid.org/`.